REDUCING CLUSTER POWER CONSUMPTION BY DYNAMICALLY SUSPENDING IDLE NODES

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Brian Michael Oppenheim

June 2010

COMMITTEE MEMBERSHIP

TITLE:	Reducing Cluster Power Consumption by Dynamically Suspending Idle Nodes
AUTHOR:	Brian Michael Oppenheim
DATE SUBMITTED:	June 2010
COMMITTEE CHAIR:	Phillip Nico, Ph.D.
COMMITTEE MEMBER:	Christopher Lupo, Ph.D.
COMMITTEE MEMBER:	Aaron Keen, Ph.D.

Abstract

Reducing Cluster Power Consumption by Dynamically Suspending Idle Nodes

Brian Michael Oppenheim

Close to 1% of the world's electricity is consumed by computer servers [23]. Given that the increased use of electricity raises costs and damages the environment, optimizing the world's computing infrastructure for power consumption is worthwhile. This thesis is one attempt at such an optimization. In particular, I began by building a cluster of 6 Intel Atom [5] based low-power nodes to perform work analogous to data center clusters. Then, I installed a version of Hadoop [1] modified with a novel power management system on the cluster. The power management system uses different algorithms to determine when to turn off idle nodes in the cluster.

Using the experimental cluster running a modified Hadoop installation, I performed a series of experiments. These tests assessed various strategies for choosing nodes to suspend across a variety of workloads. The experiments validated that turning off idle nodes can yield power savings. While my experimental procedure caused the apparent throughput to significantly decrease, I argue that using more realistic workloads would have yielded much better throughput with slightly reduced power consumption. Additionally, my analysis of the results, show that the percentage power savings in a larger, more realistically sized cluster would be higher than shown in my experiments.

Acknowledgements

I am fortunate to have been supported by many family members, friends, and instructors. I would like to especially thank the following people:

- **Linda and Lee (My Parents)** - Very few Master's students are fortunate enough to have their parents willing and excited to join them at their defense presentation. Mom and Dad, it came as no surprise that you were the exception to the rule. For my entire life you have given me nothing less than your strongest support for everything that I have wanted to do.

- **Jason (My Brother)** - I am so lucky to have such a supportive little brother. Thank you for coming up for my defense.

- **Willie, Flo, "Papa Sarge", and Barbara (My Grandparents)** - Thank you all for your unconditional support throughout my life. The money you put aside for my education helped make it possible for me to stay at Cal Poly for an extra year to complete my Master's degree. Grandpa Willie and Grandma Barbara, although you were not able to live to see me complete my Master's thesis, I have never forgotten the support that you gave me.

- **Dr. Phil Nico (My Thesis Advisor)** - Thank you for all of your time helping me to prepare my thesis. I am thankful that you gave me guidance and advice while allowing me to keep ownership of my project. I am also glad that you pushed me during the last couple of weeks of my project to ensure that my paper clearly described the entirety of my efforts.

- **Dr. Chris Lupo (Member of My Thesis Committee)** - I appreciate all of your advice in setting up a proper study of power consumption. I am

also grateful for you lending me your power and temperature measurement devices.

- **Dr. Aaron Keen (Member of My Thesis Committee)** - Thanks for introducing me to Hadoop in your CSC-458 Parallel Computing class. The class helped inspire my decision to work on a thesis related to parallel computation.

- **Computer Science Department Faculty and Staff** - Thank you to all of the faculty members and staff in the Computer Science department that I have had the pleasure of learning from or working with. I value all of the time you have put into helping me succeed often on an informal, first-name basis. I particularly want to thank: Julie Workman, Diane Nott, Cindy Bitto, Dr. Ignatios Vakalis, Dr. Zoë Wood, Dr. Gene Fisher, and Dr. Alexander Dekhtyar.

- **Prentice Wongvibulsin (My Research Partner)** - Thank you for all of the brainstorming sessions and hardware help. I also appreciate being able to split the cost of hardware with you, allowing us both to do our projects on larger clusters.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Nearly 1% of the world's electricity was consumed by computer servers in 2007 [23]. Between 2000 and 2005, data center energy consumption increased 16.7% worldwide. Given the rapidly increasing demand for computing infrastructure, there is reason to believe that data center energy usage has continued to increase. Since the heightened use of electricity increases costs and harms the environment, optimizing the world's computing infrastructure for power consumption is worthwhile. Recognizing the negative consequences of increased power consumption, companies try to balance their need for expanded computing power with being energy efficient [13].

This thesis is one attempt at a power efficiency optimization, making use of a technique called Vary On Vary Off (VOVO for short) [15, 28]. I began by building a cluster of 6 Intel Atom [5] based low-power nodes to perform work analogous to commercial production data center clusters. Then, I installed Hadoop [1] on the cluster. Hadoop is a leading open-source clustering framework that is, to a large degree, written to solve MapReduce problems. Hadoop is often used for large-scale calculations that can be run faster over a cluster of nodes. Common

Hadoop tasks include web indexing for search engines and mining information from large data sets [1]. Given its popularity in large clusters, often on the order of thousands of nodes, it is worth considering Hadoop's power efficiency.

Previous research has explored running Hadoop on a cluster of low-power hardware. Other works (as described in Chapter 2) have investigated different ways of optimizing Hadoop to run in a more power efficient manner. Building on such ideas, my thesis serves to minimize the power consumed by Hadoop clusters of low-power nodes. Using the VOVO technique, I aimed to dynamically turn off idle nodes in the cluster. In order to accomplish a VOVO-based power management system in Hadoop, I developed an extensible power management framework. Within this framework, I created a few simple scheduling policies that use different algorithms to determine when to suspend nodes in the cluster.

Using the experimental cluster running a Hadoop installation modified with my power management system, I performed a series of experiments. These tests assessed various strategies for choosing nodes to suspend across a variety of workloads. The experiments validated that turning off idle nodes can yield power savings. In fact, the tests with the heaviest load showed that turning off idle nodes can save at least 13.3% power. In my analysis of such results, I argue that the percentage power savings in a larger, more realistically sized cluster would be higher. I also discuss the effects my system has on throughput.

1.1 This Paper

The rest of this paper is structured as follows. Chapter 2 presents the necessary background information and a survey of related works. Chapter 3 uses the introduction chapter, the background information, and related works to clearly

state my thesis question. Chapter 4 is where I discuss the framework implementation. Chapter 5 talks about the tests that I performed on the experimental cluster, the results, and analysis thereof. Chapter 6 gives concluding remarks that tie the work together and highlight the key achievements of this thesis. Chapter 7 suggests ways in which the work could be extended and improved.

Chapter 2

Background and Related Works

There are a couple of different areas of research that I bring together in this thesis. In this chapter, I provide background information about these areas. I also highlight recent work done on topics similar to this work.

2.1 Energy Efficient Computing

As I briefly discussed in the Introduction (Chapter 1), energy efficient computing has become an important area of study for major corporations. The goal of these efforts is to minimize both the monetary and environmental costs of large data centers. By one estimate, server machines globally consume over 0.8% of the world's electricity [23]. In a separate paper, Koomey estimated that total server energy consumption doubled from 2000 to 2005 [24]. Some have tried to address this issue under the assumption that computers consume energy in proportion to the amount of work they do. This would mean that under close to 0 CPU utilization, one would expect to see close to zero power consumption [19]. This model is attractive for its simplicity and since it matches the human intuition

that it should take more power to do more work.

Works such as [19] have shown that proportional model does not accurately represent actual power consumption. Instead, power consumption is made up of 2 components, one fixed, and one that varies with workload. The fixed component of power consumption is typically a relatively large portion of the machine's total usage. This large chunk represents fans, mechanical drives, LEDs, and other devices that are on when the computer is turned on. The other part of this model is power consumption that is proportional to CPU usage. While both the older and newer models of power consumption agree with our intuition that peak power consumption happens under the highest possible load, they differ significantly in the power consumption during idle periods. In the newer model, the fixed power component shows us that even close-to-idle machines meaningfully contribute to total power consumption.

One study quantified the fixed plus variable power consumption result, showing that, while idle, an 8-core Xenon processor consumes around 60% of the power that it consumes at full utilization of all cores [30]. The idle power consumption percentage becomes even more important when you consider that server utilization in data centers is only around 20% to 30% [27]. Assume for a moment that the work was partitioned as to have 25% of the nodes at 100% utilization with the rest idle. This would lead to a power total of 25% fully loaded machines $*$ 100% consumption $+$ 75% idle machines $*$ 60% consumption $=$ 7000 power units. If the idle nodes were turned off, the power total would be 25% fully loaded machines$*$ 100% consumption $=$ 2500 power units or 35% of the original power consumption (a savings of 65%).

Other research supports the figures found in [27]. In particular, a pair of Google engineers studied the CPU utilization of over 5,000 of Google's servers

over a period of about 6 months. Their study showed that the target machines spent around 75% of the time at less than 50% CPU utilization. They argue that this result is not surprising because of the way internet services are provisioned to ensure maximum throughput even when traffic to the servers is high [13]. In light of the newer power consumption model, the utilization statistic raises an important question that deserves consideration. If most machines spend more than a majority of their time doing little work (and consuming non-negligible amounts of power), should they even be on? Stated another way, is it possible to move load around in a computing system to decrease the amount of machines that need to be turned on?

If nodes were to be arbitrarily turned off or removed, the cluster would accept a performance penalty during peak traffic periods. This suggests that a more flexible approach must be considered. Specifically, this implies that one way to get more optimal power efficiency is dynamically turning machines on and off in such a way as to keep powered-on nodes at high utilization. This technique, independently proposed by [15] and [28] has come to be known as Vary On Vary Off (VOVO for short) [21].

2.2 MapReduce

MapReduce is a programming model developed by Jeffrey Dean and Sanjay Ghemawat at Google. In this section, I discuss the theory behind the model as well as some implementations that have been written to solve MapReduce problems.

2.2.1 Purpose and Theory

The model decomposes certain programming tasks into two main steps, each operating on key/value pairs. First, a *map* phase takes each input key/value pair and outputs a set of key/value pairs. These intermediate key/value pairs are then processed with the *reduce* phase which combines the values of all pairs having the same key. This results in a single value output for each key.

In their paper, Dean and Ghemawat say that such a decomposition is natural for many real-world problems, citing web indexing and a few other examples as support. The problem decomposition used in MapReduce is perfectly suited to be used in a parallelized environment. In particular, parallelism can be exploited in two ways. First, within the map and reduce phases, data can often be partitioned into independent chunks that can be operated on in parallel. Second, as the map phase begins to generate output, the reduce phase can start combining incoming results rather than waiting for the map phase to first complete [20].

To illustrate how MapReduce works, I will use the example of counting the number of occurrences of each word in a given text. There are numerous ways of performing this task using MapReduce, however, I only describe the process shown in Figure 2.1. First, the raw text input is split up by sentence. Each sentence is put as the value part of a key-value pair. Since we do not need a key, it is null in each input pair. The Map function is then evaluated for each sentence. For each unique word in the given sentence, the Map function generates a pair with the word as the key and the number of occurrences of the word in the sentence as the value. In the reduce phase, all pairs with the same key (representing the same word) are collected. The value part of each group of keys is then summed to give the total number of occurrences of the word. A pair with

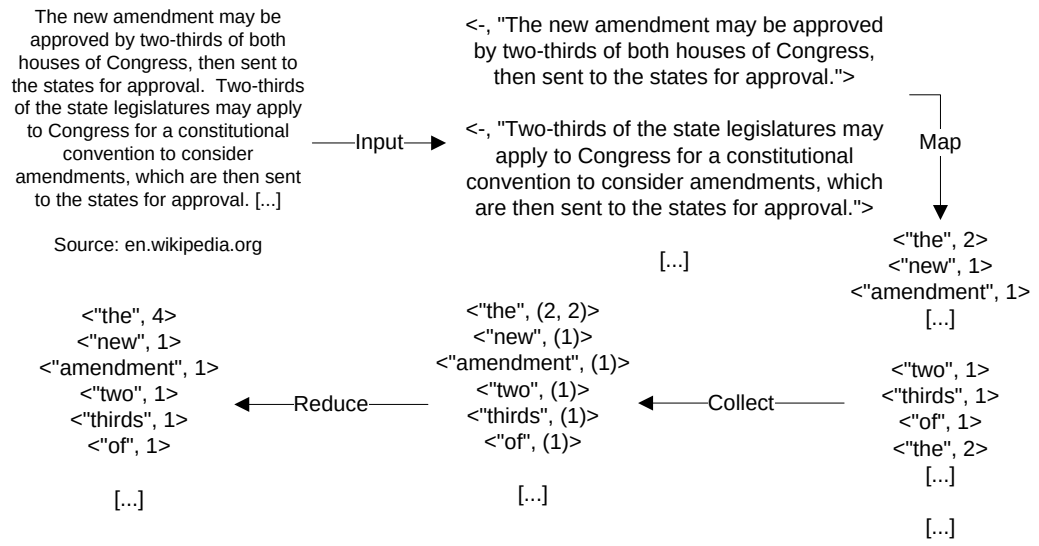

Figure 2.1: A diagram showing a sample run of the given word count MapReduce implementation.

the word as the key and the sum of occurrences as the value is outputted.

2.2.2 Implementations

Dean and Ghemawat's initial MapReduce paper describes their implementation of a MapReduce system [20]. Since the publication of that initial paper, several alternate implementations of MapReduce have been developed.

General Purpose

Hadoop, which will be the focus of this paper, is the quintessential open-source MapReduce implementation. Developed by the Apache Foundation, the Hadoop framework is written in Java [1]. Hadoop is further discussed in Section 2.3. Twister is another popular Java-based MapReduce system. The Twister website describes the framework as a lightweight implementation of Hadoop that performs well on iterative jobs [11]. Other implementations of MapReduce lever-

age the power of existing database technologies to run MapReduce processes. In particular, AsterData's SQL-MapReduce combines SQL syntax with the MapReduce programming model to directly run MapReduce operations inside an existing database [9]. Another MapReduce implementation, called Greenplum, allows MapReduce actions to happen on top of existing databases. Greenplum is built to support large-scale systems. For instance, the popular website, LinkedIn, uses Greenplum [4].

Implementations for Specialized Hardware

The implementations of MapReduce that I've presented so far run on standard general-computing machines. Specialized MapReduce implementations, however, have been written for specific hardware. For example, [6] and [7] have been written for the cell processor platform and GPUs respectively. These implementations strive to abstract away the complexity of specialized hardware, making it easier for the developer to efficiently utilize the device's parallel capacity.

2.3 Hadoop

Hadoop is an open-source, general-purpose computing cluster framework with an emphasis on MapReduce computation. In this section, I discuss the framework and point out features that are important to understanding my thesis.

2.3.1 Purpose and History

The history of Hadoop can be traced back to the Apache Software Foundation's Lucene [2] project. Lucene is an open-source API that powers some text-

based search engines. Specifically, it provides indexing and searching features for the document set under consideration by the search engine. Using Lucene, developers created a new product called Nutch [3]. In addition to having the index provided by Lucene, Nutch gives search engine developers features such as a web crawler. Nutch also provides search engine developers extra information about their index such as a link graph. As Nutch continued to grow, developers realized that MapReduce [20] was a highly efficient way of performing the indexing operations that Nutch performed on webpages. In order to take advantage of MapReduce, Nutch developers created new infrastructure for Nutch, which became what is now Hadoop. Since then, the Hadoop project has grown to include a distributed file system, a data warehouse component, a distributed locking mechanism, a data serialization system, and a few other subprojects [1].

For the purposes of this thesis, the focus will be on the main components that are used in a Hadoop cluster, the MapReduce system and the distributed file system. The file system component allows users to save data files in a distributed, replicated manner. This data is then accessed by MapReduce jobs which each run over numerous nodes in the cluster. Typically such jobs are written as Java applications that simply implement the `Mapper` and `Reducer` interfaces defined by the Hadoop API. Since a Java application is not always feasible or appropriate, Hadoop also supports the use of arbitrary binaries as map functions and as reduce functions. This feature is called Hadoop Streaming.

2.3.2 Implementation and Architecture

The Hadoop core framework, written in Java, is made up of two main components, the distributed file system and the MapReduce component. Following

the master/slave distributed design pattern, both the distributed file system and MapReduce parts of the framework run 2 types of daemons, client and server. On the system's single server node, the distributed file system and MapReduce components each run a server daemon. The server daemons for the distributed file system and MapReduce components are called `NameNode` and `JobTracker` respectively. Likewise, in each client node of the system, both of the components run a client daemon. The distributed file system client daemon is called a `DataNode` and the MapReduce client daemon is called a `TaskTracker`. These labels are shown in the Hadoop system diagram given in Figure 2.2. [1]

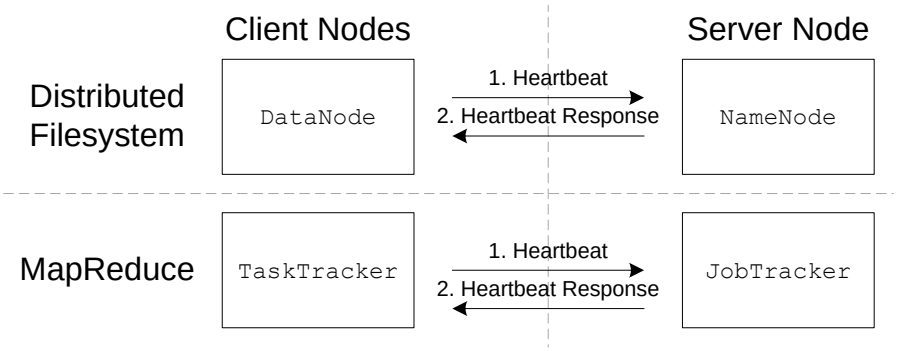

Figure 2.2: A system diagram of the Hadoop MapReduce and distributed file system client and server nodes.

Most (but not all) communication in a Hadoop cluster is initiated by client nodes to a server node. This communication is done over a simple RPC mechanism that is part of Hadoop's core implementation. Since the clients initiate the communication, the server nodes can only give instructions to the client nodes when the client nodes initiate contact. Additionally, the server is only aware that a given client is alive and healthy when that client decides to inform the server of such. In order for the server daemons to be able to give out work and be kept up-to-date as to client node status, both client daemons (`TaskTracker` and `DataNode`) running on each node regularly send a "heartbeat" RPC to their coun-

terpart on the server node (`JobTracker` and `NameNode`). This communication is generally sent about every 2 seconds, but is configurable by the user. Each server daemon inspects the status update in each heartbeat it receives and replies with information on what the sender should be doing. This communication process is shown graphically in Figure 2.2. [1]

Hadoop is a system that encompasses multiple pieces of hardware and processes running on thousands of nodes. Therefore, Hadoop has many methods of detecting, handling, and cleaning up after various system errors. These error handling mechanisms form a safety net protecting the system from subtle issues that may be introduced by the power management system developed in this thesis. For instance, if a node becomes stuck in a suspended state, Hadoop will eventually treat it as it would any other dead node and would then perform necessary cleanup operations. If for some reason a node is suspended while it is performing a task of a MapReduce job, causing the task to not complete, the `JobTracker` will notice. The `JobTracker` would handle this situation as it would with any other failed task attempt. Specifically, the `JobTracker` would try to restart the task, likely on a different node.

Distributed File system Component

In order to maintain data reliability and to allow data to be quickly accessed during MapReduce computations, Hadoop employs its own distributed file system. The system is built such that files are divided into discrete blocks that are each stored on multiple cluster nodes. Not suprisingly, it is known as "Hadoop Distributed File System" or HDFS for short. The HDFS contains a single main server known as the `NameNode`. The `NameNode` contains all information about the files in the file system including their names, permission, ownership, and the

locations of their data blocks. The data blocks are distributed amongst the client HDFS nodes, known as `DataNode`s. The size and number of replicas of each data block are configurable both on the level of the file system and individual files. As a rack-aware system, Hadoop uses a replication policy that balances the speed benefits of having data on the same rack with having data on other racks for safety. [1]

MapReduce Component

The MapReduce component is responsible for managing everything needed to successfully execute a MapReduce program on a Hadoop cluster. First, configuration options, program binaries, and input data, collectively called a *Job*, are provided to the system's server, called the `JobTracker`. The `JobTracker` then inspects the Job configuration and reviews the current state of the cluster's resources. Based on all of this information, the `JobTracker` determines how to split up and schedule the `Job` on the cluster. That is, how the input data should be partitioned and assigned to available nodes for the map phase of the computation. Each discrete chunk of work assigned to a node is called a `Task`.

The `JobTracker` then sends each map `Task` to the `TaskTracker` running on the cluster node that will execute the `Task`. The `JobTracker` also determines how reduce `Task`s will be executed on the cluster. These `Task`s get the outputs from the map `Task`s and perform the reduce operations on them. [1] Most often, the outputs of the reduce phase, and many times the Map-phase inputs as well, are stored on the Hadoop distributed file system. This inter-communication between the MapReduce and HDFS components necessitates careful planning when modifying either component.

2.4 Low Power Clusters

With the advent of low power chips, attempts have been made to use them while applying existing clustering techniques. The main motivation for this work is to see if these clusters of low power machines can achieve better performance per unit power than currently available computational systems. A team at CMU showed in [33] that clusters of low power machines can be as much as six times more power efficient (as measured by queries per joule) than a typical machine. They accomplish this by using an array of machines each with an embedded 500MHz processor and compact flash storage. Their research also showed that there is a difference in the appropriate considerations for seek-bound (loads that access various parts of disk) and scan-bound (loads that generally access the data on the disc sequentially) work loads.

Others have made clusters of nodes using more typical off-the-shelf hardware. In [31], a variant of an Intel Atom chip is used for the CPU instead of the embedded processor. Instead of using compact flash cards as the storage mechanism, their nodes use solid state drives. This decision allows them to speed up sequential read throughput by an order of magnitude with constant power consumption. They argue that their cluster of so-called Amdahl nodes gives five times the throughput on data-intensive workloads over current state-of-the-art computing clusters. They also argue that this balance between low power consumption and high throughput is the way to achieve optimal performance per watt [31]. This result matches well with the conclusions of the energy-efficient computing research discussed in Section 2.1.

Another team provides numerical results to support this conclusion. This team built a system called Gordon that used flash storage and data centric pro-

gramming systems. In their tests, Gordon beat disk-basked clusters by 1.5x and was able to do so at 2.5x more performance per watt [14].

2.5 Hadoop Power Efficiency

There have been other attempts to look at Hadoop's power efficiency using a variety of different approaches. The most general look came in a paper from a team at Stanford. Their main point is that Hadoop's lack of power efficiency stems from the idle time experienced by cluster nodes. They also suggest and briefly evaluate strategies, such as new data placement algorithms, that allow idle machines to be turned off [25].

UC Berkeley researchers have been particularly active in the search for power efficient Hadoop clusters. In one of their tech reports, the Berkeley researchers describe a method by which they take actual traces of cluster execution and generate statistically-based workloads. These workloads are played back in order to measure the power consumption of clusters [16]. In a different technical report, the Berkeley team developed general models for calculating a Hadoop cluster's power efficiency (and by extension general MapReduce power consumption). These models are parameterized based on attributes such as number of nodes, number of workers, job duration, idle power, active power, replication, and expected failure rates. From the models they developed, the authors argue that optimizing for energy consumption in a Hadoop cluster is equivalent to traditional performance optimizations [17].

Earlier this year (February 2010), a South Korean team took what they claim to be the first look at running Hadoop on low-power hardware. They found that while there was a signficant performance hit on the low-power hardware with

respect to the standard commercial hardware, the low-power machines exhibited 113 times better performance per watt. In order to reduce power consumption even further, the work looks at ways that data can be intelligently moved between nodes so as to consolidate work, allowing for some machines to be temporarily suspended [22].

Finally, a team of Swiss computer scientists modified Hadoop's scheduling and block distribution algorithms to be more power efficient. Their work was motivated by their findings that systems with sleeping nodes incur an unreasonable performance penalty due to the unavailability of the data housed on the sleeping nodes. In order to combat the performance issues, the researchers modified the Hadoop MapReduce and distributed file system components to make the components actively aware of power management decisions. This additional information is fed into the algorithms that schedule jobs and select which nodes to save data on. The authors say that their experiments show that these modifications do make positive progress in reducing energy consumption by Hadoop clusters [32].

2.6 SPECpower_ssj2008

With the growing interest in scientific studies of power consumption, researchers needed a way to standardize their measurements. The use of standardized experimental procedures allows researchers to draw meaningful comparisons within and between studies. In order to address this need, the Standard Performance Evaluation Corporation (SPEC for short) developed the first-ever benchmark for measuring server power consumption compared to system performance. This benchmark is called "SPECpower_ssj2008" [8]. The benchmark

runs a Java application that runs tasks that consume varying amounts of CPU time (10%, 20%, 20%, etc.) The runs at different throughput percentages are seperated by varying intervals of time.

The benchmark mandates adherence to specific rules to ensure consistent meaning between experiments. First, while experimenters may view and edit the benchmark source code, they must run the SPECpower binary compiled by SPEC. Additionally, the benchmark specification restricts certain optimizations. In particular, optimizations can only be written if they optimize the general system rather than only targeting a known part of the benchmark. Finally, the benchmark only allows certain measurement devices to be used for official tests. These devices must also conform to configuration rules, again to ensure consistent results.

Chapter 3

My Idea

Having explained all of the background information, I can state the main question to be explored in my thesis: How can commodity, low-power hardware be used to run a Hadoop cluster in the most power efficient manner? In an attempt to answer to that question, I make 2 research contributions through this paper.

First, I present new VOVO-based framework for power management of a Hadoop cluster. In the proposed framework, I aim to minimally change existing Hadoop code. At the same time, I work towards a flexible framework design, allowing for easy customization of power scheduling behavior.

Second, I performed a set of experiments on a Hadoop cluster made of commodity, low-power hardware. These experiments used the framework I developed in order to test various ways of determining when to turn off nodes in the cluster. The goal of the tests were twofold. First, the experiments are collectively used to determine the viability of using VOVO on a cluster of low-power Hadoop nodes. Second, the tests determined the relative power savings of different scheduling

algorithms. In this paper, scheduling algorithms are the methods by which nodes are selected to be turned off.

Chapter 4

Implementation

In order to investigate my thesis question, I developed a novel VOVO-based power management system for Hadoop. This involved both development of new classes and some modification to existing Hadoop code. I explain how the implementation is designed and how it operates with current Hadoop infrastructure in this chapter.

4.1 Goals

Before designing and building the power management system, I set goals and constraints for the system. First, I prioritized keeping as much existing code of Hadoop unchanged as possible. This implied that I should keep Hadoop mostly unaware of the power management scheme. In addition to keeping the code and functionality changes of Hadoop to a minimum, there were other quantities that I wanted to optimize. For example, it was important to have minimal loss of throughput, minimal power consumption, and minimal system complexity. This all needed to be done in a way that did not introduce errors or significant limita-

tions onto the Hadoop system. Additionally, a very important design goal was to make the system easily modifiable and extensible. In order to work towards that goal, I needed to use a modular design that would allow for different components to be plugged in and out of the system. Modularity serves not only in the interests of extensibility and possible future work, but it also allows for experiments to be easily performed with different power management policies.

4.2 Overview

My implementation of a power management system is a combination of newly written components and some small modifications to Hadoop. I performed all development and testing on the Hadoop release from February 26, 2010, known as version 0.20.2-dev. In describing the behavior of the system that I developed, I will refer to the step numbers shown in Figure 4.1. In addition to being included in this section's narrative, I briefly summarize the labeled steps in figure 4.2.

The VOVO-based power management system that I created is controlled by a server called the `NodePowerManager` (further described in Section 4.3). This server coordinates with all four of the core Hadoop daemons (`TaskTracker`, `JobTracker`, `NameNode`, and `DataNode`) to turn on and off cluster nodes. Additionally, the `NodePowerManager` is built to be operated with a scheduler that dictates when on and off operations should take place on specific cluster nodes (see Section 4.5). Part of the new communication between the core Hadoop system and the power management system involves a pair of complementary new mechanisms called *freezing* and *thawing* (see Subsection 4.4.3). Additionally, I created a new concept called pseudoheartbeats (see Subsection 4.4.1) to simulate Hadoop heartbeat communications with suspended nodes.

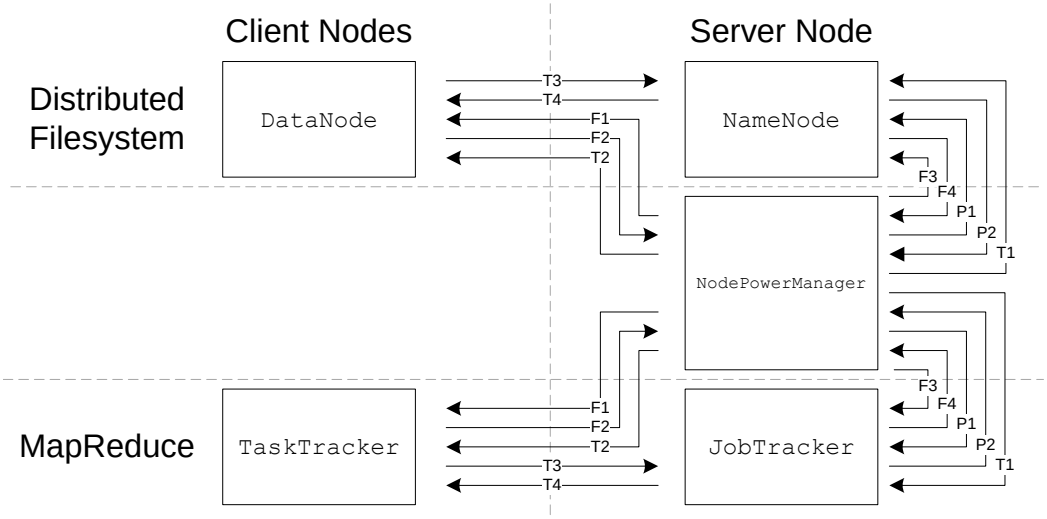

Figure 4.1: A system diagram showing Hadoop with the power management system. Each step is further explained in Figure 4.2. F=Freeze/Sleep, P=Pseudoheartbeat, T=Thaw/Wake

F1: Request that the node freeze.
F2: Tells if the freeze request succeeded or not (i.e. if the node is doing any work or not).
F3: Request the last received heartbeat for the node being put to sleep.
F4: Returns the heartbeat requested in step **F3**.
P1: Send the pseudoheartbeats.
P2: Receive the pseudoheartbeat responses and check if they contain work to be done.
T1: Thaw the client nodes.
T2: The most recent pseudoheartbeat replies are sent back to their sender to be sent to the client in step **T5**.
T3: The newly awoken daemons send their first heartbeat since waking up.
T4: The daemons send the response sent to it in step **T3**.

Figure 4.2: A step-by-step description of the Hadoop implementation modified to include the power management system. The steps numbers refer to the diagram in Figure 4.1.

4.3 Core Components

The main component of the Hadoop power management systemn is a Java server called the `NodePowerManager`. The `NodePowerManager` is responsible for keeping track of the power state of the cluster machines (i.e. suspended or awake). The server also performs the necessary operations to suspend cluster nodes or to wake them up. Most requests to change the power state of a node come either from a scheduler (discussed in 4.5) or via Hadoop's built-in RPC mechanism. As I describe later, in Subsection 4.4.1, a wake action can also be triggered when a Hadoop server assigns work to a sleeping node (step **P2** in figure 4.1).

Due to the distributed nature of the system, I gave the server a locking mechanism that prevents multiple power actions to be taken on the same node at the same time. For instance, while a request to wake up a node is being serviced, the server cannot process a request to turn the same node off. While I provide a brief description of the "sleep" and "wake" operations in this section, it is worth noting that the actual implementations are slightly more complex to ensure proper synchronization and robust error handling.

In order to manually request that a node be put to sleep or be waken, I created a small utility program called `NodePowerChanger`. This program is operated exclusively by command line arguments. These parameters tell which operation (sleep or wake) to perform and which nodes to perform the action on. The program itself simply sends RPCs to the `NodePowerManager` to perform the desired action(s). The utility is mostly to be used for debugging and testing purposes. It could also be used as an external means to perform power management tasks on the cluster in environments where RPCs are not feasible (such as from a non-Java program).

4.3.1 Sleep Operation

The "sleep" operation begins with steps **F1** and **F2** in figure 4.1. These steps represent performing checks that make sure that the target node is not doing any work. This logic also instructs the node to temporarily stop accepting any new work, a step called *freezing* the node (see Subsection 4.4.3). Next the pseudoheartbeat process (as described in Subsection 4.4.1) is initiated. After these internal operations are performed, the `NodePowerManager` establishes an `ssh` connection to the target node. Once connected, it executes Ubuntu's `pm-suspend` command to put the machine into the suspend state. Upon verification, by ICMP pings, that the node has been shut down, the pseudoheartbeats begin (see Subsection 4.4.1).

4.3.2 Wake Operation

The "wake" operation starts by `NodePowerManager` using the `wakeonlan` [26] utility to wake the suspended node. Upon the node becoming responsive again (as verified by ICMP pings), the `NodePowerManager` tells the client node, via RPC, to begin accepting work again (step **T1**). As the complimentary action to the hold placed on the node during the "suspend" phase, it is referred to as *thawing* the node (see Subsection 4.4.3). Additionally, the `NodePowerManager` stops sending the pseudoheartbeats after thawing the node (see Subsection 4.4.1).

4.4 Modifications to Hadoop

While a main goal of my implementation was to hide the details of the power management system from Hadoop, there were a few places where modifications to

the existing code were unavoidable. I detail the changes that I made to Hadoop in this section.

4.4.1 Faking It: Pseudoheartbeats

The only way that the Hadoop servers know that a given client node is alive and functioning is via the periodic heartbeats that the client sends. When suspended, client nodes cannot send the heartbeat. Given the goal of minimizing changes needed to Hadoop for this power management system, I needed an external solution. This issue was addressed by implementing a technique called "pseudoheartbeat"s.

Immediately prior to shutting down a node, the `NodePowerManager` contacts both of the server node daemons. In this contact, the `NodePowerManager` gets a copy of the last heartbeat they received from their corresponding client daemon on the node being shut down (steps **F3** and **F4**). These heartbeats represent each of the client daemon's most recent status update. Once the node is turned off, the `NodePowerManager` begins to send this most recent heartbeat as if it were the client daemon (step **P1**). This heartbeat will stay valid since the node's status cannot change while it is asleep. If any of the heartbeat replies (step **P2**) for a node contain work to be done, the wake up process is initiated.

Before it issues the command to wake up a sleeping node, the `NodePowerManager` uses a special RPC to send back to the appropriate server daemons the last heartbeat responses that it received (step **T2**). This special RPC tells the server that it should reply with that response the next time it hears from the specified node. Thus, once the target node is woken up and attempts to issue its next heartbeat (step **T3**), it will receive (in step **T4**) the response that was received on its behalf

by the `NodePowerManager`.

4.4.2 Preventing Sleep-Talking

Nodes in the cluster that have been put to sleep will not be able to accept incoming connections. Thus, before communicating with a client node, data senders need to have some way to be sure that the receiving node is awake. To achieve this, I first located all code in both the server and client parts of the file system and MapReduce components of Hadoop that initiate an outbound connection. At each of these locations, I added an RPC call to the `NodePowerManager` to wake the target host. I designed this RPC to be lightweight so that it adds minimal delay to the send process in the common case that the target node is already awake.

4.4.3 Freezing and Thawing

Due to the complex synchronization that must take place between the servers and clients of both the MapReduce and distributed file system components, there needs to be a mechanism to temporarily stop a client daemon from accepting work to be done. More specifically, from the time the suspend procedure is initiated to the time the node is woken back up, it is important that the node not start any new work. Otherwise it could accept work then immediately shut off, leaving the requestor waiting for the work to be completed. In order to alleviate this issue, I retrofitted the client daemons with RPC functions that allow callers to "freeze" or "thaw" them. Freezing temporarily pauses the daemon from accepting work while thawing acts as its counterpart, un-pausing the server.

4.5 Schedulers

As I described in the system overview, I wrote the power management system with extensibility and future development in mind. In particular, I wrote the `NodePowerManager` so that the user may specify any scheduling policy that they wish. For the purposes of this work, a scheduling policy is any algorithm that determines when to suspend or wake nodes in the cluster. In order to implement such a policy, the user need only implement a simple interface called `NodePowerScheduler`. The interface specifies a single method, `schedule`, that takes as an argument the `NodePowerManager` instance that it should use to perform scheduling operations. Users are free to build implementations of `NodePowerScheduler` that use any information that they determine is relevant to scheduling decisions. In my simple implementations, the schedulers only know whether or not a given machine is turned on. As discussed in the future work (Chapter 7), more powerful schedulers may be developed that use more information about the cluster's state.

Alongside the `NodePowerScheduler` interface, I authored several implementations of the interface. Each implementation specifies a different possible scheduling policy. The first, very trivial, scheduler that I implemented was one that does nothing. This scheduler can be used for manual testing as well as for establishing a baseline power consumption figure.

The simplest non-trivial scheduler that I implemented regularly attempts to turn off all nodes that are not performing work. The frequency of this action is configurable. In order to prevent nodes from rapidly being turned on and turned off, I added an option that enforces a minimum awake period. That is, the scheduler will not try to shutdown a node until it has seen it awake for at

least some configurable amount of time. In order to try to balance power savings with quick data availability, I developed a round-robin scheduler. This scheduler behaves by keeping a list of all nodes in the cluster and turning them off one-by-one on configurable intervals.

Finally, I wrote a scheduler that makes decisions at random. This scheduler randomly picks one node to shutdown on configurable random intervals. While it seems unlikely that the random scheduler will provide the most power savings, using it will provide a useful baseline figure. That is, other more intelligent scheduling choices can be compared to the random scheduler to see if doing something "intelligent" is worth the effort.

4.6 Known Issues

The system that I developed works well as a proof-of-concept project to illustrate the ideas of this thesis. However, there are several instances during its development and testing where speed and ease of development took priority over concerns that would be important in a production environment. These issues would need to be addressed in order for this system to be ready for real-world deployment. While addressing some of these issues might require some effort, none should have any meaningful impact on the power consumption of the system.

4.6.1 Scalability

In order to send the pseudoheartbeats, the power manager currently starts two threads per sleeping node. Each of these threads require a network socket to communicate with the appropriate Hadoop server daemon. Both of these

resources, threads and network connections, are limited in numbers. While these system-dependent limits do not come into play in a small cluster, such as the experimental system used in this thesis, they must be considered in clusters containing thousands of nodes. This issue would likely be solved by using thread and resource pools.

4.6.2 Response to Cluster Changes

Nodes in a Hadoop cluster can be removed manually or as a result of an error condition. Due to time limitations during the coding phase, I was not able to make the power management system aware of nodes that have been removed from the cluster. As a consequence of this simplification, once the `NodePowerManager` has errors contacting a node, it may not be able to do so for the rest of the system's execution. This is true even if the node becomes reachable again.

4.6.3 Security

The shutdown method that I wrote in the `NodePowerManager` used the `pm-suspend` command on client nodes to place the machine into suspend mode. This command requires superuser privileges to execute. In the interest of time, the simplest way to accomplish this was to give the Hadoop user `sudo` access with no password. This is a huge security risk since a malicious MapReduce program could easily gain root access to any machine. With a little added effort, it would be possible to secure this suspension procedure.

The other security vulnerability is the lack of authentication needed for power management network communication. For instance, the `wakeonlan` command can be executed by any user on any machine with access to the cluster's network

[26]. Without a proper network set-up, an unauthorized user could wake up a machine in the cluster. Additionally, a similar vulnerability exists in the RPC system used by the power management code. On an unfirewalled network, any user could communicate with the RPC mechanism and change power states or freeze/thaw nodes. However, it is important to note that security as a whole in Hadoop is relatively weak. In many places it is a work in progress. Techniques for securing RPCs in other Hadoop components should be applicable for securing them in the power management system. Such techniques are beyond the scope of this work.

4.6.4 Power Management

There are several issues that prevent the power management controls from being robust in a production environment. First, the system relies on the wake-on-lan protocol for bringing nodes up from suspend state. This protocol requires special configuration on the network card and BIOS of each node. Additionally, wake-on-lan packets are sent over the UDP [29] protocol [26]. UDP is, by definition, an unreliable protocol. This causes 2 problems against its practicality. One of these problems is that there is no guarantee that any given wake-on-lan packet will reach its destination. Thus, wake-on-lan packets may need to be sent multiple times. Secondly, there is no built-in acknowledgement of success. Therefore, to be sure that a machine has been awakened, an external mechanism such as ICMP pings must be used. As an alternative to wake-on-lan, a daughter card or other external hardware solution could be used to wake nodes.

In addition to the wake-on-lan reliability issue, the power management system suffers from machine identification issues that could be cumbersome or even error-

prone in a production environment. First, MAC addresses are required in order to use the wake-on-lan protocol. In order for the `NodePowerManager` to have the MAC addresses of the nodes in the cluster, the addresses had to be hardcoded. Alternatively, they could have been supplied via a text file or other external storage mechanism. Both of these approaches are cumbersome and very inflexible. Besides MAC address issues, the system suffers from IP address issues as well. These issues stem from Hadoop's inconsistent identification of cluster nodes. In some cases, nodes are referred to by their DNS hostname and in other cases they are referred to by IP address. This poses a problem if a machine's IP address changes after the cluster is started up. Such an issue could arise in network configurations that rely on DHCP for instance or as a result of poorly coordinated system maintenance.

4.6.5 Synchronization

Synchronization can be an issue in a standalone program or in simple server-client applications. It becomes a much bigger issue in a more complex system such as Hadoop. In particular, the `NodePowerManager` must coordinate with client and server parts of both the HDFS and MapReduce components. While I implemented the `NodePowerManager` with such concurrency issues in mind, the server could benefit from a careful analysis looking for bug-causing race conditions.

4.6.6 Client Web Interfaces

Each client and server daemon in the MapReduce and HDFS components expose an HTTP web interface. The interface gives status information about each

of the daemons. This includes data such as file system usage and MapReduce job status as well as access to files stored in the HDFS. Since the web servers are run as part of the daemon process on which they report, they become inaccessible when the host machine is placed into suspend mode. Thus, administrators cannot use the web interfaces of daemons on nodes which have been shut down. As a work around, an administrator could use the `NodePowerChanger` utility to manually wake up a suspended node in order to gain access to its web interfaces. Alternatively, a smart proxy server could be developed that would wake a sleeping node if any pages on any of its web interfaces are requested.

Chapter 5

Experimentation

In this chapter, I present the experiments that I performed in order to validate the power management framework. I also give the results of the experiments and analysis of the results.

5.1 Experimental Set-Up

My experimental set-up involved several components. I describe each of those components in this section.

5.1.1 Cluster

I performed the tests on a cluster of homogeneous commodity hardware costing approximately $150 per node. Each of the 6 nodes had the hardware as specified in Figure 5.2. A picture of the experimental cluster is shown in Figure 5.1. The figure also shows the reasons that each piece of hardware was selected for the cluster. Generally, I picked parts with the goal of balancing low power

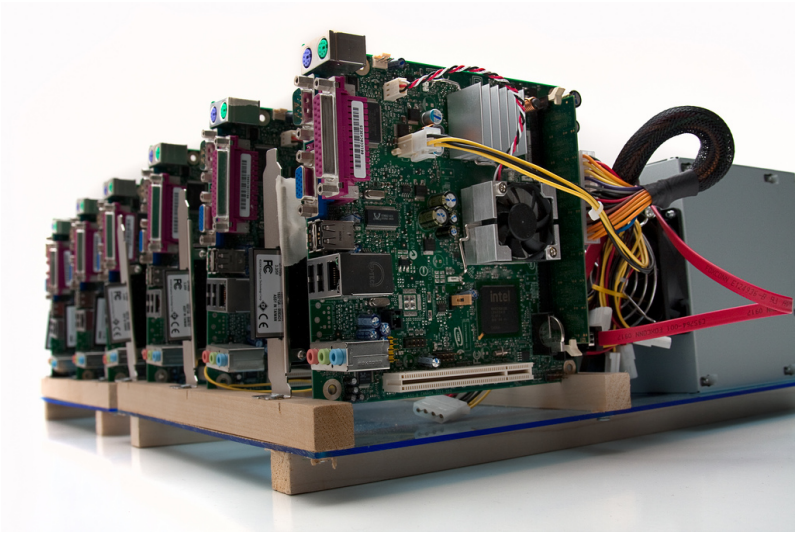

Photo Credit: Prentice Wongvibulsin (http://www.prenticew.com)

Figure 5.1: A picture of the 6 nodes in the experimental cluster.

consumption and reasonable cost.

Before running the experiments on my modified Hadoop implementation, I used one node from the cluster to test the relationship between CPU load and power consumption. This test was meant to provide a quick snapshot of the nodes' power consumption properties. Therefore, to account for slight synchronization issues between the timing of the CPU load and power consumption measurements, each data point represents the moving average of 5 data points (approximately 5 seconds). The results of these tests are shown in figure 5.3. The graph shows a constant term of about 52.05 watts. This means that, when idle, each node in the cluster consumes approximately 52.05 watts. The graph also shows a slope of about 3.06. This indicates that for every 1% of CPU load, power consumption increases by 0.0306 watts.

These results match the conclusions from the background work papers that I discussed earlier. In particular, my nodes consume 52.05 watts when idle and

- **Intel Atom processor 330** (Dual Core @ 1.6GHz; L1 Cache: 1MB; L2 Cache: 2x512KB; 533MHz FSB) - I selected the atom CPU due to its position as a leading off-the-shelf low-power processor [5].
- **Intel BOX94GCLF2 Motherboard** - I chose this board since it provides a small form-factor and comes with the atom chip embedded.
- **Rendition by Crucial** (1GB of DDR2 667 SDRAM PC2 3500) - The chosen RAM was an economical option meeting the motherboard's memory requirements.
- **Roswell RV300 Power Suppy** (300W Maximum Power; 70% minimum efficiency) - The Roswell power supply's 70% minimum efficiency rating was very high with respect to its low price.
- **Kingston 4GB Compact Flash Memory Card (3.3V)** - In order to save power as compared to a mechanical hard drive, I used a compact flash card as the primary storage device. I chose this card for its speed and its use of the lower CF card power specification (3.3V instead of 5V).
- **SYBA SD-ADA40001 SATA II to Compact Flash Adapter** (Jumper Choice: 3.3V or 5V) - Since the motherboards did not have a compact flash card reader built in, I purchased this device to put a CF card reader on one of the board's SATA interfaces.

Figure 5.2: Specification of nodes in the experimental cluster.

55.11 watts at 100% load. Thus, when idle, the nodes consume about 94.4% of the power that they consume when fully loaded. As noted in the background works section, this suggests that optimization techniques that involve turning nodes off have significant potential for bringing power savings.

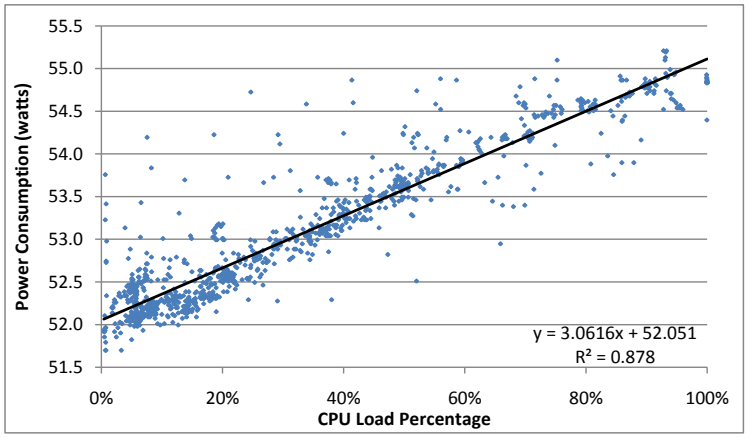

Figure 5.3: A graph showing the effects of CPU load on single node power consumption.

I used a giggabit switch to connect the nodes of the cluster. Specifically, I used Netgear's ProSafe Gigabit 8 Port Ethernet Switch (model GS108). The switch consumes approximately 3 watts of electricity.

5.1.2 Measurement Tools

I used several existing tools to measure and log various experimental quantities. I also developed a couple of new tools for this thesis. In this subsection, I talk about all existing and new measurement tools used in the experiments.

Power Measurement

In order to measure the power consumed by the cluster, I used a GW Instek GPM-8212. The GPM-8212, shown in figure 5.4, is a popular, relatively inexpen-

sive device that measures the amperage, voltage, frequency, and watts on a given load. In addition to the front display that visually shows the current readings, the device has an RS232 port that provides access to the same information. On PC systems, the port is read over a COM port.

Figure 5.4: A picture of the GW Instek GPM-8212 power meter.

The manufacturer provides basic software for interfacing with the device. However, the program only gives the same live data that can be seen on the front panel. Since the important quantity in the experiments is the total power consumption, I needed to log the measurements over time. I accomplished such logging by developing a Java program written to communicate with the device on regular intervals and record its readings. I also released the software as an open source project at `http://code.google.com/p/java-gpm8212`.

Thermal Measurement

As required by the SPEC power benchmark (discussed in 2.6), I needed to take regular temperature measurements during the experiment. To take these measurements, I used a TEMPer USB device. A picture of the device is shown in figure 5.1.2 [10]. The unit came with software that logs the temperature reading on regular, configurable intervals.

Figure 5.5: A picture of the TEMPer USB Thermometer.

CPU Usage Measurement

To gain better insight into the cluster's behavior, I logged the CPU usage of each node throughout most of the tests. I started with the shell script from [18]. I updated the script to regularly output log entries along with a timestamp. The script can be seen in Appendix A.

5.1.3 Use of SPECpower_ssj2008 Guidelines

Since the tests in this thesis are about measuring the power consumption of a cluster across different conditions in software, I did not run the actual SPECpower benchmark. However, I followed many of its experimental guidelines to make my results as meaningful as possible [8]. Specifically, I did the following:

- **Power Measurement** - I used the GW Instek GPM-8212 with its default configuration. Additionally, the load was less than 10A as required by the benchmark's rules for this device.
- **Thermal Measurement** - I used an unsupported device, the TEMPer USB thermometer to measure temperature. The device logged samples at a rate of 1 per second, well in excess of the benchmark's 4 per minute requirement. The device's accuracy of ±1°C was outside the required ±0.5°C accuracy. However, the temperatures measured during all experiments were above the 20°C minimum temperature requirement.

5.2 Tests and Results

The goal of the experiments was to thoroughly test the system under varying conditions. I tested the system using every combination of four scheduling algorithms (one that did not turn off any nodes) running on four different sized workloads. I ran each of these 16 tests for 10 minutes, starting each test at the moment that Hadoop finished initializing the MapReduce component. Throughout the entire test, I measured and logged power consumption and ambient temperature. Additionally, I logged CPU usage on all machines during most tests. After each run, I removed any data generated by the test from the cluster to restore it to a clean state.

5.2.1 Workloads

The main distribution of Hadoop comes with several example cluster programs packaged together as `hadoop-examples`. The following example programs were used to make the workloads:

- `pi` - Uses the monte carlo method to estimate pi. In my experiments, I used 150 maps, each with a sample size of 100,000.
- `teragen` - Generates input data for the Terabyte Sort Benchmark. For my tests, the program was set to generate data sets of approximately 50MB in size.
- `terasort` - Performs the sorting component of the Terabyte Sort Benchmark.
- `teravalidate` - Checks the output of a terasort run for being in sorted

order, outputting any incorrect orderings.

- `wordcount` - Counts the frequency of words in the given input files. As input, I used four copies each of plain-text versions of Alice in Wonderland, Huckleberry Finn, Pride and Prejudice, and The Adventures of Sherlock Holmes.

Using the programs above, I formulated 4 workloads. While the complete scripts used to generate the workloads are given in Appendix B, the following gives a brief description of each:

- **No Load** - No jobs run on the cluster.
- **Light** - Jobs are run one-at-a-time with at least 35-54 seconds between the conclusion of one job and the start of the next.
- **Medium** - Jobs are run one-at-a-time with 25-32 seconds between the conclusion of one job and the start of the next.
- **Heavy Load** - More jobs are submitted to the cluster than it can process during the test. This keeps cluster utilization as close as possible to 100%.

5.2.2 Scheduling Algorithms

I used the following scheduling algorithms to test my framework:

- **None** - No nodes are ever turned off.
- **All Off** - Every 10 seconds, the scheduler calls for all nodes to be shut down.

- **Round Robin** - Every 10 seconds, the scheduler picks one node to be shut down. The nodes are picked in round-robin order.

- **Random** - On random intervals, between 5 and 30 seconds, one node is randomly picked to be shut down.

5.2.3 Results

The results of the experiments on the cluster are summarized in Table 5.1 and given as percentage improvements in Table 5.2. Additionally, the results are shown graphically in Figures 5.6, 5.7, and 5.8. While the three graphs represent the same dataset, they each highlight different features of the results. The 3D graph (Figure 5.6) gives the best high-level overview of the entire set of results. The bar graph in Figure 5.7 gives a good comparison of each scheduling algorithm's relative performance across the different workloads. In contrast, the bar graph in Figure 5.8 shows how the scheduling algorithms performed relative to each other on each workload.

Scheduling Policy	No Load	Light Load	Medium Load	Heavy Load
None	0.0439	0.0468	0.0452	0.0482
Random	0.0190	0.0414	0.0433	0.0463
Round-Robin	0.0147	0.0409	0.0433	0.0453
All Off	0.0151	0.0407	0.0386	0.0418

Table 5.1: Results of experiments on the cluster. Each value shows the total power consumed (in killowatt hours) by the cluster in each test.

Scheduling Policy	No Load	Light Load	Medium Load	Heavy Load
Random	56.7%	11.5%	4.2%	3.9%
Round-Robin	66.5%	12.6%	4.2%	6.0%
All Off	65.6%	13.0%	14.6%	13.3%

Table 5.2: Results of the experiments, expressed as percentage improvements over the appropriate baseline test.

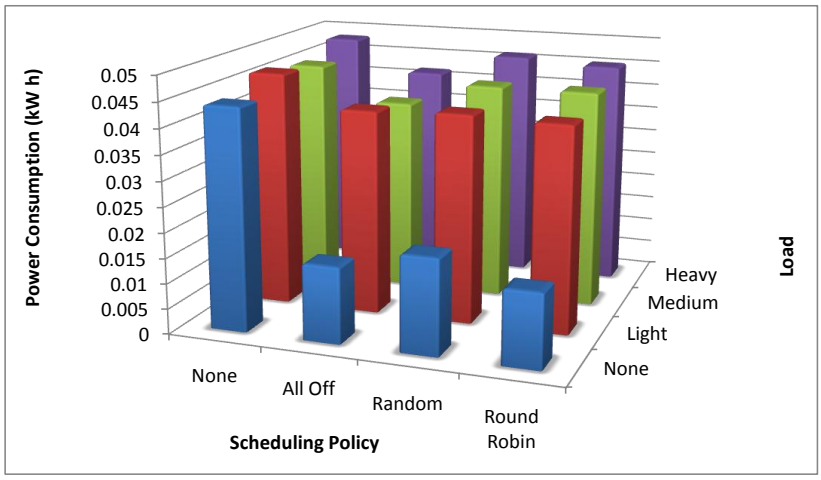

Figure 5.6: A graph showing the results of the experiments on the cluster. This is a graphical representation of Table 5.1.

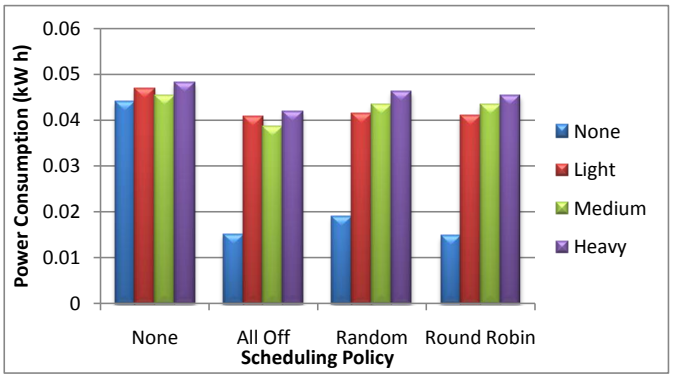

Figure 5.7: A chart of experimental results by scheduling policy.

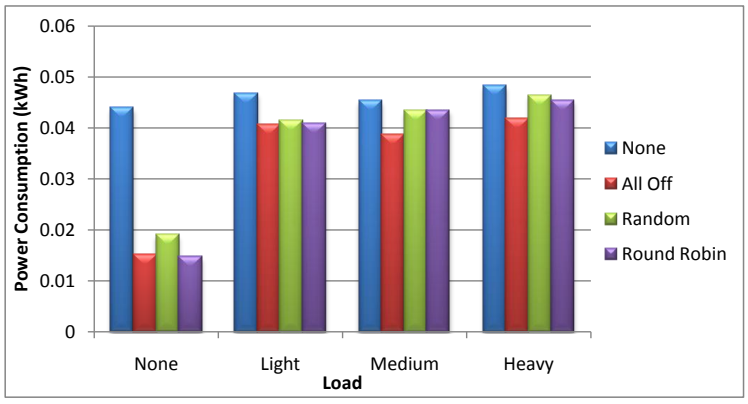

Figure 5.8: A chart of experimental results by load.

Unfortunately, due to the structure of the experiments, these power savings came at the cost of throughput. While the experiments were not set up in a way that would easily allow the throughput to be measured, Hadoop's `JobTracker` logs do provide some insight into how much work was accomplished. Although the amount of work done in each `Job` was not exactly equal, the unit of throughput I used was the number of completed `Job`s. I computed a throughput percentage for each test that used the `NodePowerManager` and ran the light, medium, or heavy workload. This percentage represented the portion of `Job`s completed in the test as compared to the baseline test for the same workload. These throughput figures are shown in Table 5.3. Analysis of these results, explaining why the throughput in most of the tests was not closer to 100% is provided in Section 5.3.4.

Scheduling Policy	Light Load	Medium Load	Heavy Load
Random	80.0%	99.0%	66.7%
Round-Robin	80.0%	83.3%	75.0%
All Off	71.8%	65.7%	60.0%

Table 5.3: A table showing the percentage of completed `Job`s compared to not using the `NodePowerManager`.

5.3 Analysis

The results show that my work does save power over an unmodified Hadoop cluster. In this section, I take a closer look at the results and analyze their implications.

5.3.1 A Little Goes a Long Way

Under each of the four workloads, the simplistic VOVO-based power management strategies saved power over leaving all nodes turned on. Specifically, the

best VOVO-based power management policy in the no load, light load, medium load, and heavy load tests saved 66.5%, 13.0%, 14.6%, and 13.3% power respectively. It is important to look at these results in the context of the scheduling algorithms that produced them. In particular, the experiments used simplistic scheduling strategies for determining when to shut down nodes and still saved a notable amount of power.

To better illustrate the savings, I will present some calculations regarding the cost of energy in a theoretical data center. Consider a California data center running 2,000 of the same nodes as used in the experiments. According to the United States Department of Energy, the February 2010 average retail price of electricity to commercial customers in California was \$0.1213 per kW·h [12]. Using this energy rate, Table 5.4 shows the power usage and cost assuming energy usage proportional to the experimental results.

		No Load	Light Load	Medium Load	Heavy Load
Power	Baseline	769,128	819,936	791,904	844,464
	Modified	257,544	713,064	676,272	732,336
	Savings	511,584	106,872	115,632	112,128
Cost	Baseline	\$93,295.23	\$99,458.24	\$96,057.96	\$102,433.48
	Modified	\$31,240.09	\$86,494.66	\$82,031.79	\$88,832.36
	Savings	\$62,055.14	\$12,963.57	\$14,026.16	\$13,607.13

Table 5.4: A table of energy savings for a theoretical California data center running 2,000 of the experimental nodes. Power measurements are in kW·h.

5.3.2 Why All-Off Didn't Sweep the Results

Under each of the workloads except for no-load, the all-off strategy performed best. A subtle difference accounts for all-off not having the largest power savings in the no load case. The all-off scheduler enforces a minimum time that nodes

must be awake before shutting them down. However, the round robin scheduler has no such requirement. Thus, the round robin scheduler begins turning nodes off immediately when the cluster is started up while the all-off does not until after the delay has passed. In a larger cluster running for longer than 10 minutes, I would expect the all-off strategy to regain its position as the leading algorithm among the ones that I tested.

5.3.3 Potential for Further Power Savings

There are several reasons that suggest that my techniques have the potential to save an even greater percentage of power than they did in my experiments. I explain a couple of such reasons in this subsection.

Flapping: Idleness Detection Issues

Before my system shuts down a node in the cluster, it verifies that the node is not performing any work for either the file system or MapReduce component. This check for work in each component of the target node represents only an immediate boolean property, whether or not the node is currently doing any work. This definition is important since the check does not look ahead at work that may be easily predicted to be coming. For instance, the node could have just finished a `Task` and is about to start a new one. Alternatively, the node could have completed a `Task` and is waiting to send the result to the appropriate location. In these situations and others, a node could be turned off and then quickly turned back on.

Having a node turn off and quickly back on can cause extra power usage in two different ways. First, I'll consider the likely case that the node was very briefly

not doing work. In this situation, extra power is consumed bringing down and bringing back up the node over what would have been consumed had the power off operation been delayed until after the second unit of work was completed. Secondly, the best-performing policy, all-off, will not call for a node to be turned off unless it has been turned on for a configurable amount of time. Thus, if after being put to sleep the node is waken to complete a quick `Task`, it will not be eligible to be turned off for some time. During this gap, the node consumes power that it would not have consumed had it been allowed to complete the small, second task before being shut down.

Cluster Size: Bigger is Better

When viewing the results, it is important to consider the small size of the cluster (6 nodes). In particular, constant power costs such as the single master node and networking equipment are estimated to comprise at least 17% of the total power. As the number of nodes is increased this percentage would approach zero. In a cluster with thousands of nodes, the value would be negligible.

In order to consider how the results might look in a large-scale cluster, I propose applying a simplistic adjustment technique based on the power consumption figures given in Subsection 5.1.1. First, I assume that the power consumption of the giggabit switch stays constant at about 3 watts. Second, I assume that the master node consumes the lower bound power consumption of 52.05 watts (as shown in Figure 5.3). Therefore, using these assumptions, I can reduce all measurements by the killowatt hours consumed when 55.05 watts are drawn for 10 minutes. With this adjustment, potential power savings is around 24% more than each reported result. The results of this adjustment are shown in Table 5.5.

Scheduling Policy	No Load	Light Load	Medium Load	Heavy Load
Random	58.1%	11.8%	4.3%	4.0%
Round-Robin	68.1%	12.9%	4.3%	6.2%
All Off	67.2%	13.3%	15.0%	13.6%

Table 5.5: Potential results of the experiments on a much larger cluster, expressed as percentage improvements over the appropriate baseline test.

After applying the adjustment, the best VOVO-based power management policy in the no load, light load, medium load, and heavy load tests could save 68.1%, 13.3%, 15.0%, and 13.6% power respectively.

5.3.4 Explaining Lost Throughput

As shown in Table 5.3, the experiments showed a notable decrease in throughput in tests with the `NodePowerManager`. At first glance, these results are concerning since this thesis aims to power off idle hardware. However, on closer examination of the workload generation scripts (shown in Appendix B), we can find a significant source of throughput loss. We see that `Jobs` are submitted in a way that does not correctly mimic real server load. In particular, `Jobs` should have been submitted at a rate that is constant over each test. Instead, the rate that some `Jobs` were submitted in each test depended on the execution time of certain previous `Jobs`. While the root cause was the same, this issue came into play in the light and medium workloads differently than it did in the heavy workload.

Light and Medium Workloads

In the light and medium workloads, sleep commands of varying amounts were used to moderate the influx of `Jobs` into the cluster. These sleeps came between

the end of one `Job`'s execution and the start of the next `Job`. Thus, on the tests that used the `NodePowerManager`, the delay introduced by sleep and wake operations pushed the submittal of future `Job`s back. Therefore, less `Job`s were able to execute during the 10 minute tests. In a realistic workload, the delay between submittal of `Job`s should not depend on the completion of other `Job`s.

The problem in the light and medium workloads can be best illustrated with an example. In the medium workload test with the round robin scheduler, the throughput was 83.3%. Assuming that the remaining 16.7% of the workload would take a proportional amount of time to the rest of the test, $\frac{10 \text{ minutes}}{83.3\%} - 10 \text{ minutes} = 120$ seconds would be needed to complete the rest of the workload. However, during that test, there was 115 seconds of sleep time between `Job`s. Thus, if `Job`s were submitted at the same intervals as the baseline test, instead of delayed by sleep and wake operations, most of the remaining 16.7% of the workload would have executed in the 10 minute test.

Heavy Workload

In the heavy workload, the rate of `Job` submissions did not turn out to be sufficient to keep the cluster fully loaded as expected. In fact, the cluster experienced idle periods between `Job`s that varied in each test. This behavior can be explained by the burst-based nature of `Job` submissions that the workload script prescribed (notice the interleaving of background and foreground tasks in the script). The correct behavior for this test would have been to repeatedly submit a `Job` and then briefly wait (significantly less than the expected execution time of the `Job`). This would have ensured that there would not be idle periods in the cluster while new `Job`s were processed.

5.3.5 Applicability to Other Hardware

The VOVO-based power management system that I developed relies on the standard standby power mode and wake-on-lan capability found on most commercial off the shelf motherboards. Therefore, the framework should operate on clusters made of hardware other than what was used in my experiments. The potential power savings on alternative hardware would depend on a couple of factors. First and foremost, the idle power consumption is an important factor. Specifically, the more power the system consumes while idle, the more opportunity the framework has to save power. Additionally, the amount of time it takes to put a node into suspend and the time it takes to wake it back up are determining factors. More power is saved when these operations take less time.

Chapter 6

Conclusions

My thesis sought to contribute new knowledge about running Hadoop on a lowe-power cluster in a power efficient manner. Recognizing the implications of previous work that modeled system power consumption, I developed a simple framework for imposing a Variable On Variable Off (VOVO) power management strategy to a Hadoop cluster. In order to validate the framework as well as to assess different scheduling policies, I performed experiments. These experiments tested the power consumption of the experimental cluster using four different scheduling policies each tested using four different workloads.

My experiments showed that VOVO is, at the very least, a strategy which does decrease power consumption on a low-power Hadoop cluster over a system with no power management strategy. Of the scheduling algorithms I investigated, the all-off strategy was most often the best choice. This policy regularly attempts to turn off all nodes in the cluster. The results showed, however, that the power saved came with a loss of throughput. However, my analysis argued that the loss in throughput was mostly attributed to workload scripts that did not adequately represent the patterns found in real workloads.

In addition to presenting the results of the experiment, I provided analysis that aimed to draw broader conclusions from my specific experiments. First, I proposed an adjustment to my experimental results to predict how the results would scale from the tested six node cluster up to a cluster of thousands of nodes. Additionally, I explained why the results on the specific experimental hardware are likely valid in other configurations.

Chapter 7

Future Work

My thesis contributes a simple framework for implementing VOVO-based power management schemes on a Hadoop cluster. My work also provides experimental results that justify the framework's viability as a power management strategy on a cluster of low-power nodes. While the framework and experiments are each meaningful research contributions in their own right, there are several improvements that could be made. There is also other opportunities for further research.

7.1 Testing

One limitation of this work is the testing performed. Specifically, the experimental cluster was made up of only 6 nodes. Real production Hadoop clusters are built using thousands of nodes. While it seems reasonable that the results and ideas given in this thesis would scale up to larger clusters, actual experiments would need to be performed to confirm this theory.

Additionally, as discussed in Subsection 5.3.4, the workloads used in the tests, did not adequately represent real workloads. This issue caused a significant drop in throughput when using the `NodePowerManager`. A new set of tests should be performed in which the intervals between `Job` submissions are constant across tests. This comes in contrast to the experiments performed in this work that had dynamic spacing between `Job` submissions based on factors surrounding the actual execution of the test.

7.2 Hadoop Modifications

For this project, I had a goal of making minimal modifications to existing Hadoop code. Researchers authoring other works, such as [32], have taken the opposite approach. Their research is based off of modifying the way in which the Hadoop file system determines where to store file block replicas. A future investigation that tied the two ideas together would be worthwhile. More specifically, the research would be focussed on creating a scheduling algorithm for this paper's power management system that would operate on a cluster using the replication policy from [32]. Additionally, the Hadoop `JobTracker` could be modified to prefer assigning jobs to nodes that are already turned on over assigning a job to a suspended node. Such an optimization has the potential to save both time and power.

7.3 Using Cluster State

In addition to being unaware of the block allocation policy, the scheduling algorithms that I wrote and discussed in this work barely used information about

the state of the cluster. The only information that they did use was a simple binary indication of whether or not a given node was doing work. It seems likely that the power management scheme could be improved by using forecasting techniques based on recorded cluster load history. In addition, the scheduling algorithm might benefit from knowledge of file access frequency and block replica locations. This information would allow the scheduler to prioritize nodes to be shut off based on their expected performance and power impact in terms of file system access.

7.4 Alternate Hardware

The hardware used in my thesis was purposely picked because of its low power consumption and its affordability on a college-student's budget. The power management system design considerations were based off of recent work such as [33] that investigates clusters of low power machines. However, it is too early to tell if clusters based on low power hardware are going to gain popularity. Thus, it is important to see how the developed power management scheme performs on a cluster of more traditional nodes. Similarly, there may be interesting results to be found by using a hybrid cluster. That is, a cluster made up partially of low power hardware and partially of more traditional hardware. Benefits in this environment would likely require changes to the power management logic to be aware of the trade offs of turning off nodes running different hardware.

7.5 Interrupting Sleep Operations

As I mentioned in Section 4.3, I used a locking mechanism to prevent multiple power operations from being taken on the same node. The enforcement of the lock forces sleep operations to fully complete before a wake operation can begin. In situations such as what is described in Subsection 5.3.3, the system could benefit from being able to interrupt a sleep operation before it fully completes.

7.6 Known Issues from Section 4.6

Finally, there are the issues brought up in Section 4.6 that would prevent the proposed framework from being a usable option in a production setting. For the sake of brevity, here is a short overview of the issues presented in that section:

- **Scalability** - In a large cluster, the power manager may run out of threads and/or network connections.
- **Response to Cluster Changes** - The power manager assumes that once a node is added to the system, it is never removed.
- **Security** - None of the RPCs sent or received by the power management system are authenticated nor limited by source/destination.
- **Power Management** - The methods for turning nodes on and off are not robust enough to handle large, dynamic cluster configurations.
- **Synchronization** - The system needs a complete analysis to verify that there are no race conditions. This is a concern due to the number of separate processes and machines that must be dealt with in a particular order.

- **Client Web Interfaces** - The web interface exposed by each Hadoop daemon becomes unaccessible when the node is suspended.

Bibliography

[1] Apache Hadoop. http://wiki.apache.org/hadoop/.

[2] Apache Lucene Project. http://wiki.apache.org/lucene-java.

[3] Apache Nutch Project. http://wiki.apache.org/nutch.

[4] Greenplum MapReduce. http://greenplum.com/technology/mapreduce.

[5] Intel Atom Processor. http://www.intel.com/technology/atom/.

[6] Mapreduce on cell. http://mapreduce-cell.sourceforge.net/.

[7] Mars: A MapReduce Framework on Graphics Processors. http://www.cse.ust.hk/gpuqp/Mars.html.

[8] SPEC power ssj2008. http://www.spec.org/power_ssj2008/.

[9] SQL-MapReduce. http://www.asterdata.com/resources/mapreduce.php.

[10] TEMPer USB Thermometer. http://www.usbfever.com/index_eproduct_view.php?products_id=257.

[11] Twister: Iterative MapReduce. http://www.iterativemapreduce.org/.

[12] Average Retail Price of Electricity to Ultimate Customers by End-Use Sector, by State. http://www.eia.doe.gov/electricity/epm/table5_6_b.html, May 2010.

[13] L. Barroso and U. Hölzle. The case for energy-proportional computing. *Computer*, pages 33–37, 2007.

[14] A. Caulfield, L. Grupp, and S. Swanson. Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications. *ACM SIGPLAN Notices*, 44(3):217–228, 2009.

[15] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, pages 103–116. ACM, 2001.

[16] Y. Chen, A. S. Ganapathi, A. Fox, R. H. Katz, and D. A. Patterson. Statistical Workloads for Energy Efficient MapReduce. Technical Report UCB/EECS-2010-6, EECS Department, University of California, Berkeley, Jan 2010.

[17] Y. Chen, L. Keys, and R. H. Katz. Towards Energy Efficient MapReduce. Technical Report UCB/EECS-2009-109, EECS Department, University of California, Berkeley, Aug 2009.

[18] P. Colby. Calculating CPU Usage from /proc/stat. http://colby.id.au/node/39, October 2008.

[19] S. Dawson-Haggerty, A. Krioukov, and D. Culler. Power Optimization–a Reality Check. Technical report, University of California at Berkeley, October 2009.

[20] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, San Francisco, CA, October 2004. Google, Inc., USENIX.

[21] E. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. *Lecture Notes in Computer Science*, pages 179–196, 2003.

[22] H. Kim, D. Shin, Y. Yu, H. Eom, and H. Yeom. Towards Energy Proportional Cloud for Data Processing Frameworks. In *Proceedings of First USENIX Workshop on Sustainable Information Technology (SustainIT '10)*, San Jose, CA, February 2010.

[23] J. Koomey. Estimating total power consumption by servers in the US and the world. Technical report, Lawrence Berkeley National Laboratory, February 2007.

[24] J. Koomey. Worldwide electricity used in data centers. *Environmental Research Letters*, 3(3):034008, September 2008.

[25] J. Leverich and C. Kozyrakis. On the Energy (In) efficiency of Hadoop Clusters. In *Proceedings of Workshop on Power Aware Computing and Systems (HotPower '09)*, October 2009.

[26] P. Lieberman. White Paper: Wake on LAN Technology, Revision 2. http://www.liebsoft.com/pdfs/Wake_On_LAN.pdf, June 2006.

[27] D. Meisner, B. T. Gold, and T. F. Wenisch. PowerNap: eliminating server idle power. *SIGPLAN Notices*, 44(3):205–216, 2009.

[28] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. Load balancing and

unbalancing for power and performance in cluster-based systems. In *Workshop on Compilers and Operating Systems for Low Power*, volume 180, pages 182–195. Citeseer, 2001.

[29] J. Postel. RFC768 - User Datagram Protocol. http://www.ietf.org/rfc/rfc768.txt, August 1980.

[30] S. Rivoire, P. Ranganathan, and C. Kozyrakis. A comparison of high-level full-system power models. *HotPower*, 2008.

[31] A. Szalay, G. Bell, H. Huang, A. Terzis, and A. White. Low-Power Amdahl-Balanced Blades for Data Intensive Computing. In *Proceedings of 22nd ACM Symposium on Operating Systems Principles*. ACM, October 2009.

[32] N. Vasic, M. Barisits, V. Salzgeber, and D. Kostic. Making Cluster Applications Energy-Aware. June 2009.

[33] V. Vasudevan, J. Franklin, D. Andersen, A. Phanishayee, L. Tan, M. Kaminsky, and I. Moraru. FAWNdamentally Power-efficient Clusters. In *Proceedings of HotOS XII*, Monte Verita, Switzerland, May 2009.

Appendix A

CPU Usage Logging Script

```
#!/bin/bash
# by Paul Colby (http://colby.id.au), no rights reserved ;)
#
# Modified by Brian Oppenheim (http://www.brianopp.com).

PREV_TOTAL=0
PREV_IDLE=0

while true; do
  CPU=(`cat /proc/stat | grep '^cpu '`) # Get the total CPU statistics.
  unset CPU[0]                          # Discard the "cpu" prefix.
  IDLE=${CPU[4]}                        # Get the idle CPU time.

  # Calculate the total CPU time.
  TOTAL=0
  for VALUE in "${CPU[@]}"; do
    let "TOTAL=$TOTAL+$VALUE"
  done

  # Calculate the CPU usage since we last checked.
  let "DIFF_IDLE=$IDLE-$PREV_IDLE"
  let "DIFF_TOTAL=$TOTAL-$PREV_TOTAL"
  let "DIFF_USAGE=1000*($DIFF_TOTAL-$DIFF_IDLE)/$DIFF_TOTAL"
  let "DIFF_USAGE_UNITS=$DIFF_USAGE/10"
  let "DIFF_USAGE_DECIMAL=$DIFF_USAGE%10"
  echo `date +%s`, $DIFF_USAGE_UNITS.$DIFF_USAGE_DECIMAL
```

```
  # Remember the total and idle CPU times for the next check.
  PREV_TOTAL="$TOTAL"
  PREV_IDLE="$IDLE"

  # Wait before checking again.
  sleep 1
done
```

Appendix B

Workloads

B.1 Light Workload

```
#!/bin/sh

bin/hadoop jar hadoop-0.20.1-examples.jar pi 150 100000
sleep 46s
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output1
sleep 54s
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output2
sleep 46s
bin/hadoop jar hadoop-0.20.1-examples.jar teragen 1000000 terainput
sleep 49s
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output3
sleep 44s
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output4
sleep 49s
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output8
sleep 48s
bin/hadoop jar hadoop-0.20.1-examples.jar teragen 1000000 terainput2
sleep 53s
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output6
sleep 42s
bin/hadoop jar hadoop-0.20.1-examples.jar terasort terainput teraoutput
sleep 46s
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output5
```

```
sleep 48s
bin/hadoop jar hadoop-0.20.1-examples.jar teravalidate terainput teradiff1
sleep 46s
bin/hadoop jar hadoop-0.20.1-examples.jar terasort terainput2 teraoutput2
sleep 42s
bin/hadoop jar hadoop-0.20.1-examples.jar pi 150 100000
sleep 48s
bin/hadoop jar hadoop-0.20.1-examples.jar teravalidate terainput2 teradiff2
sleep 35s
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output7
sleep 41s
bin/hadoop jar hadoop-0.20.1-examples.jar teravalidate terainput2 teradiff3
sleep 40s
bin/hadoop jar hadoop-0.20.1-examples.jar teravalidate terainput teradiff4
```

B.2 Medium Workload

```
#!/bin/sh

bin/hadoop jar hadoop-0.20.1-examples.jar pi 150 100000
sleep 27s
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output1
sleep 29s
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output2
sleep 32s
bin/hadoop jar hadoop-0.20.1-examples.jar teragen 1000000 terainput
sleep 27s
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output3
sleep 29s
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output4
sleep 30s
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output8
sleep 28s
bin/hadoop jar hadoop-0.20.1-examples.jar teragen 1000000 terainput2
sleep 29s
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output6
sleep 28s
bin/hadoop jar hadoop-0.20.1-examples.jar terasort terainput teraoutput
sleep 26s
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output5
sleep 28s
```

```
bin/hadoop jar hadoop-0.20.1-examples.jar teravalidate terainput teradiff1
sleep 26s
bin/hadoop jar hadoop-0.20.1-examples.jar terasort terainput2 teraoutput2
sleep 27s
bin/hadoop jar hadoop-0.20.1-examples.jar pi 150 100000
sleep 31s
bin/hadoop jar hadoop-0.20.1-examples.jar teravalidate terainput2 teradiff2
sleep 25s
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output7
sleep 29s
bin/hadoop jar hadoop-0.20.1-examples.jar teravalidate terainput2 teradiff3
sleep 30s
bin/hadoop jar hadoop-0.20.1-examples.jar teravalidate terainput teradiff4
```

B.3 Heavy Workload

```
#!/bin/sh

bin/hadoop jar hadoop-0.20.1-examples.jar pi 150 100000 &
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output1 &
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output2 &
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output3 &
bin/hadoop jar hadoop-0.20.1-examples.jar teragen 1000000 terainput
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output4 &
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output8 &
bin/hadoop jar hadoop-0.20.1-examples.jar teragen 1000000 terainput2
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output6 &
bin/hadoop jar hadoop-0.20.1-examples.jar terasort terainput teraoutput
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output5 &
bin/hadoop jar hadoop-0.20.1-examples.jar teravalidate terainput teradiff1 &
bin/hadoop jar hadoop-0.20.1-examples.jar terasort terainput2 teraoutput2
bin/hadoop jar hadoop-0.20.1-examples.jar pi 150 100000 &
bin/hadoop jar hadoop-0.20.1-examples.jar teravalidate terainput2 teradiff2 &
bin/hadoop jar hadoop-0.20.1-examples.jar wordcount input output7 &
bin/hadoop jar hadoop-0.20.1-examples.jar teravalidate terainput2 teradiff3 &
bin/hadoop jar hadoop-0.20.1-examples.jar teravalidate terainput teradiff4 &
```